Report from Dagstuhl Seminar 13372

Integration of Tools for Rigorous Software Construction and Analysis

Edited by

Uwe Glässer[1], Stefan Hallerstede[2], Michael Leuschel[3], and Elvinia Riccobene[4]

1 **Simon Fraser University, B.C., CA,** glaesser@cs.sfu.ca
2 **Aarhus University, DK,** stefan.hallerstede@wanadoo.fr
3 **Heinrich-Heine-Universität Düsseldorf, DE,** leuschel@cs.uni-duesseldorf.de
4 **University of Milan, IT,** elvinia.riccobene@unimi.it

Abstract

This report documents the program and the outcomes of Dagstuhl Seminar 13372 "Integration of Tools for Rigorous Software Construction and Analysis". The 32 participants came from 10 countries: Australia, Austria, Brazil, Canada, Denmark, France, Germany, Great Britain, Italy, Norway. The aim of the seminar was to bring together researchers and tool developers from different state- and machine-based formal methods communities in order to share expertise and promote the joint use of modelling tool technologies. Indeed, each of these communities – from Abstract State Machines, to B, TLA, VDM, Z – has valuable tools and technologies which would be beneficial also for the other formal approaches. Understanding and clarifying their commonalities and differences is a key factor to achieve a possible integration or integrated use of these related approaches for accomplishing, in a rigorous way, the various modelling and analysis tasks to construct reliable high quality software systems.

The working group formula offered by the Dagstuhl seminar was a fruitful way to share knowledge of the various techniques and tools – such as simulators, animators, model checkers, theorem proves – developed for the individual methods, and to constructively experiment the combined use of different approaches by means of a series of well known case studies. Participants did not arrive with well-prepared solutions, but all the modelling and integration work was directly done in Dagstuhl in a very exciting and competitive atmosphere. Some related presentations were also given on recent advances on methodologies and tools.

The seminar posed the basis for a series of future research collaborations between different, and until now closed, formal method communities. An LNCS volume will be prepared from the contributions of the participants to give the *common vision* of future methodology and tool integration.

Seminar 08.–13. September, 2013 – www.dagstuhl.de/13372
1998 ACM Subject Classification D.2.4 Software/Program Verification, F.3.1 Specifying and Verifying and Reasoning about Programs, F.4.1 Mathematical Logic, I.6.4 Model Validation and Analysis, I.6.5 Model Development, I.6.8 Types of Simulation
Keywords and phrases Applied Formal Methods, Modelling Formalisms, Modelling Tools, Abstract State Machines, B Method, Event-B, TLA$^+$, VDM, Z, Verification, Validation, Proof, Simulation, Animation, Visualisation, Model Checking, Tool Integration
Digital Object Identifier 10.4230/DagRep.3.9.74
Edited in cooperation with Hamed Yaghoubi Shahir

1 Executive Summary

Uwe Glässer
Stefan Hallerstede
Michael Leuschel
Elvinia Riccobene

Motivation

Dagstuhl Seminar 06191 had been a success in establishing the "ABZ" joint conference for the different state-based modelling communities (e.g., ASM, B, VDM, Z, TLA$^+$) with venues in London (2008), Orford, CA (2010), Pisa (2012) and Toulouse (2014). It was a first step toward bringing these communities closer together. However, the conference, although being a place where the researchers meet, does not produce in itself a significant number of collaborations across the communities. The organisers of this seminar consider such collaborations vital in order to achieve a larger impact academically and industrially.

Aims of the seminar

The seminar aims to

1. Inspire exchange and joint use of formal modelling tool technologies
2. Establish long-term cross-community collaboration
3. Work towards a common vision on formal modelling

Points 2 and 3 are particularly important for future tool developments, a common methodical foundation, and more economic use of the necessary and available resources.

Preparation

At first the organisers intended to give the participants of the seminar case studies in advance that the participants could work on prior to the seminar to showcase their methods and tools. However, a downside of this common organisational practice is that most attendees arrive at the seminar with well-prepared, polished formal models and presentations. This would have resulted in conference-style presentations, not leaving much room for cross-community group work on problems with mixed-method approaches. Thus, no substantial gain above and beyond what the "ABZ" conferences already accomplish would have been achieved. Hence, the organisers decided to take the risk not to ask for advance preparation but have all the work done collaboratively during the seminar. This was thought to create a more open atmosphere and leave room for discussion. The organisers chose candidates for case studies to be carried out during the seminar and asked the participants to explore solutions with diverse methods in small, often mixed, groups formed dynamically based on interests. A tentative schedule for the week was published prior to the seminar. It was adapted by the organisers every night, taking into account the actual progress by the work groups and feedback received in plenum discussions held every day in the late afternoon or evening.

Execution

On Sunday evening the organisers held a three hour meeting to prepare day 1 of the seminar. It was decided that, in general, evenings should be left for the participants to socialise. The case study for day 1 needed to be well-chosen to engage the participants in the seminar. It was required that

- a single problem should be treated to minimise presentation overhead necessary for explaining the model
- the problem to be solved should not be trivial but solvable within 3 hours
- the problem should not leave too much room for interpretation so that the models, methods and tools used are more readily comparable
- the problem to be solved should come with a sketch of a solution so that focus would be on the modelling activity itself and not on finding the smartest solution.

The decision was made to use the problem of "Derivation of a termination detection algorithm for distributed computations" (EWD840) by E. W. Dijkstra.

On day 1, all but one group succeeded in producing a formal model. (That group produced their model on day 2 in the after-lunch session.) At the end of day 1, the organisers felt that not enough discussion across community boundaries was taking place. This would have to be addressed in the following days. The planning for day 2, payed specific attention to this aspect. In the evening of day 1, a two-hour planning meeting among the organisers was held. It was decided that a good way of getting the different communities involved in discussions would be to reshuffle the groups of day 1 somewhat. To carry out a comparison between the methods and tools, each group would have some members that produced the original model and some "envoys" of a group that had modelled the problem in a different notation. (This turned out to work well. By lunchtime on day 2, live discussions across the community boundaries had effectively started.) On suggestion of the participants, an originally planned plenum discussion on tool integration was carried out in three groups dealing with methodology, abstract syntax and low-level integration. (The actual number of groups was decided together with all participants in the beginning of the corresponding session. Even though it may have appeared frustrating at times for some participants, the organisers thought involving all participants in some of the decision making would also improve everyone's commitment.) The organisers also started incorporating talks. This was also considered useful for breaking the routine of the seminar.

In the evening of day 2, a one hour meeting among the organisers was held to plan day 3. It was thought that the participants could be involved closer by forming new groups that should each address a problem using two different approaches and tools. The comparison would then be possible while modelling. The modelling problem chosen was the FM'99 ATM modelling challenge. Two more talks followed on day 3 and some planning for integration meetings that should be held in smaller groups on day 4. The latter were considered to be fruitful by many with a lot of common interests being announced. In the afternoon of day 3, a shorter hike provided a welcome break, as the weather did not invite for larger excursions.

On day 3 in the evening, a 30 minute meeting of the organisers was held. The plan for day 4 was mostly to tie up the open threads from the preceding days. A short wrap up of the case studies followed by presentations giving a comparison between two methods. In the evening, a first discussion of post-seminar work was held, discussing the Dagstuhl report and a joint book on "comprehensive modelling and modelling tools". In the evening of day 4, a 30 minute meeting by the organisers was held. It was decided that the morning of day 5 should be spent discussing the joint book. Uwe Glässer presented an alternative case for use in the book to start a discussion about the writing approach that should be taken. (On day 5, it was decided to keep the ATM study but improve its description.)

Outcomes and Outlook

The main outcomes of the seminar are (a) various new collaborations across community boundaries to achieve a possible integrated use of different methods and tools, and (b) concrete plans for a book on "comprehensive modelling", a step towards a common vision of the research field. An agreement has been reached with Springer Verlag to publish the post proceedings of the seminar in the State-Of-The-Art series of Lecture Notes in Computer Science. This should improve visibility of the effort started at the seminar.

The organisers seek to get funding, e.g., by way of a network of excellence, to continue the integration work and keep up the momentum achieved during the seminar. The aim will be to develop a common vision and a more coordinated research agenda where, in particular, resources for tool development could be used more efficiently in future.

2 Table of Contents

3 Overview of Talks

3.1 Defining Communication for Abstract State Machines

Egon Börger (University of Pisa, IT)

Traditional ASMs come with the following classification of locations resp. functions (for executing agents a) [1, Ch.2.2.3]:

- *controlled*: readable and writable by a,
- *monitored*: only readable (not writable) by a,
- *output*: only writable (not readable) by a,
- *shared*: readable and writable by a and some other agent(s) subject to some conflict preventing protocol.

This classification turned out to be rather useful for numerous modelling endeavours, but it comes at the price of a rather abstract (*synchronous global*) way to deal with *communication between agents*: either via output ('send') and monitored ('receive') locations or via shared ('send/receive') locations. In particular the environment steps are supposed to happen only between two (discrete global) steps of an agent.

Our goal is to define a conservative extension of ASMs (to be compatible with previous ASM modelling work) by a model of interaction between agents which a) is helpful for practical modelling where communication between agents is an issue, b) is truly concurrent (not global synchronous) and c) offers an abstract *explicit Send/Receive* mechanism between ASMs with only private local locations. The definition should not be bound (but adaptable) to any concrete message passing system or transmission protocol, reflect in a uniform way synchronous and asynchronous communication and be implementable by an appropriate plugin for CoreAsm.

We propose for further experimentation concrete definitions for abstract synchronous and asynchronous versions of *Send/Receive* communication actions as update instructions, i.e. instructions generating updates à la CoreAsm [2]. We base these definitions on an abstraction of the input pool concept in S-BPM [3]. The details can be retrieved from the slides of the talk (and hopefully later from a paper we intend to write).

References

1 E. Börger and R. F. Stärk. *Abstract State Machines. A Method for High-Level System Design and Analysis.* Springer, 2003.

2 R. Farahbod et al. *The CoreASM Project.* http://www.coreasm.org.

3 A. Fleischmann, W. Schmidt, C. Stary, S. Obermeier, and E. Börger. *Subject-Oriented Business Process Management.* Springer Open Access Book, Heidelberg, 2012. www.springer.com/978-3-642-32391-1.

3.2 Describing the ATM in a Domain Specific Language

Albert Fleischmann (Metasonic AG – Pfaffenhofen, DE)

ASM and Event B are general methods for creating models and from practitioner's point of view not easy to understand. If requirements are defined by people belonging to the application domain then it is very helpful to use an appropriate specification language. Therefore it seems useful to create domain specific languages based on formal languages like event B or ASM. An example of such a language is S-BPM. S-BPM allows describing business processes. The semantics of S-BPM is defined as an ASM. The worked out example shows a S-BPM description of the automatic teller machine which is used as a use case. Behind that model there is a corresponding ASM and therefore the model is executable which means domain people can test a specification whether it meets their intentions.

3.3 Formal Modelling Problem: Distributed Termination Detection

Uwe Glässer (Simon Fraser University – B.C., CA)

Aiming at a simple requirements description that is concise and yet meaningful as an introductory illustrative example for comparing different state-oriented formalisation methods along with methodical aspects, we have chosen the termination detection algorithm for distributed computations proposed by Dijkstra et al. in 1983 [1]. This termination detection algorithm runs on a computer network with N machines, assuming a common network topology and reliable communication mechanisms. The original description derives the algorithm in several steps, together with an invariant that demonstrates that the algorithm works as expected. We presented the problem to the seminar participants in terms of an informal description of the basic requirements for the algorithm. A question and answer session provided further clarification. Additionally, a copy of the original paper by Dijkstra et al. was made available.

Interesting challenges in formal modelling dynamic properties of this algorithm are the inherently distributed nature of the problem and the quest for finding concise and yet appropriate abstract representations of interfaces between the algorithm and its operational environment. This environment refers to the distributed computations and the communication network. After working for several hours in small groups, a number of different solutions using diverse formalisation approaches emerged and were presented and discussed in a plenary session at the end of Day 1. Despite the relative simplicity of the problem, the comparison of the various solutions revealed interesting insights into strengths and weaknesses of the underlying formal methods.

This exercise provided a good starting point for a broader discussion about comparing and combining different methods and supporting tools for software and system analysis, validation and verification in order to complement their strengths. A central question is whether one should generally aim at tight integration methods based on common semantic frameworks or rather limit to loose integration based on transformations between formal models. While tight integration may be considered ideal, greatly simplifying the combined use of several methods, the cost of integrating continuously evolving tools appears to be prohibitive, severely limiting the practicability of any such approach. These open research questions need more scientific discussion to be answered comprehensively.

References

1 E.W. Dijkstra, W.H.J. Feijen, and A.J.M. van Gasteren. Information Processing Letters 16: 217-219, 1983.

3.4 System Modelling and Analysis using ASMETA

Paolo Arcaini (University of Bergamo, IT), Angelo Gargantini (University of Bergamo, IT), and Elvinia Riccobene (University of Milan, IT)

The ASMETA (ASM mETAmodeling) framework [1] is a set of tools around the Abstract State Machines (ASMs). It supports different activities of the system development process, from specification to analysis. ASMETA has been developed [2, 3] by exploiting concepts and technologies of Model-Driven Engineering (MDE). The starting point of the development has been the *Abstract State Machine Metamodel* (AsmM) [4], an abstract syntax description of a language for ASMs. *AsmetaL* is a platform-independent concrete syntax, and *AsmetaLc* a text-to-model compiler that parses AsmetaL models.

Simple model validation can be performed by *simulating* ASM models with the simulator *AsmetaS* [5] that supports *invariant checking*, *consistent updates checking* for revealing inconsistent updates, and *random* and *interactive* simulation. A more powerful validation approach is *scenario-based validation* by the ASM validator *AsmetaV* [6] that permits to express (through the *Avalla* modeling language) execution scenarios in an algorithmic way as interaction sequences of *actions* committed by the *user*.

Model review is a validation technique aimed at determining if a model is of sufficient quality; it allows to identify defects early in the system development, reducing the cost of fixing them. The *AsmetaMA* tool [7] permits to perform *automatic* review of ASMs; it looks for typical vulnerabilities and defects a developer can introduce during the modelling activity using the ASMs.

Formal verification of ASM models is supported by the *AsmetaSMV* tool [8] that maps AsmetaL models into specifications for the model checker NuSMV. It supports the declaration of both *Computation Tree Logic* (CTL) and *Linear Temporal Logic* (LTL) formulas.

Runtime verification is a technique that allows checking whether a run of a system under scrutiny satisfies or violates a given correctness property. *CoMA* (Conformance Monitoring by Abstract State Machines) [9] is a specification-based approach (and a supporting tool) for runtime monitoring of Java software: the conformance of a Java program is checked at runtime with respect to its formal specification given in terms of ASMs.

Model-based testing aims to use models for software testing. One of its main applications consists in test case generation where test suites are automatically generated from abstract models of the system under test. The ATGT tool [10] is available for testing of ASM models; it implements a set of adequacy criteria defined for the ASMs [11] to measure the coverage achieved by a test set and determine whether sufficient testing has been performed.

References

1 The ASMETA website. http://asmeta.sourceforge.net/, 2013.

2 P. Arcaini, A. Gargantini, E. Riccobene, and P. Scandurra. A model-driven process for engineering a toolset for a formal method. *Software: Practice and Experience*, 41:155–166, 2011.

3 A. Gargantini, E. Riccobene, and P. Scandurra. Combining formal methods and mde techniques for model-driven system design and analysis. *International Journal on Advances in Software*, 3(1,2):1–18, 2010.

4 A. Gargantini, E. Riccobene, and P. Scandurra. Ten Reasons to Metamodel ASMs. In J.-R. Abrial and U. Glässer, editors, *Rigorous Methods for Software Construction and Analysis*, volume LNCS 5115, pages 33–49. Springer Verlag, 2009.

5 A. Gargantini, E. Riccobene, and P. Scandurra. A Metamodel-based Language and a Simulation Engine for Abstract State Machines. *J. Universal Computer Science*, 14(12):1949–1983, 2008.
6 A. Carioni, A. Gargantini, E. Riccobene, and P. Scandurra. A Scenario-Based Validation Language for ASMs. In *Proceedings of the 1st international conference on Abstract State Machines, B and Z (ABZ '08)*, volume LNCS 5238, pages 71–84. Springer-Verlag, 2008.
7 P. Arcaini, A. Gargantini, and E. Riccobene. Automatic Review of Abstract State Machines by Meta Property Verification. In C. Muñoz, editor, *Proceedings of the Second NASA Formal Methods Symposium (NFM 2010)*, pages 4–13. NASA, 2010.
8 P. Arcaini, A. Gargantini, and E. Riccobene. AsmetaSMV: a way to link high-level ASM models to low-level NuSMV specifications. In *Abstract State Machines, Alloy, B and Z, 2nd Int. Conference (ABZ 2010)*, volume LNCS 5977, pages 61–74. Springer, 2010.
9 P. Arcaini, A. Gargantini, and E. Riccobene. CoMA: Conformance monitoring of java programs by abstract state machines. In *2nd International Conference on Runtime Verification, San Francisco, USA, September 27–30 2011*, 2011.
10 A. Gargantini, E. Riccobene, and S. Rinzivillo. Using Spin to Generate Tests from ASM Specifications. In *Proceedings of ASM 2003*, volume LNCS 2589. Springer Verlag, 2003.
11 A. Gargantini and E. Riccobene. ASM-Based Testing: Coverage Criteria and Automatic Test Sequence Generation. *J. Universal Computer Science*, 7:262–265, 2001.

3.5 Next-preserving Branching Bisimulation

Kirsten Winter (The University of Queensland, AU)

Bisimulations are in general equivalence relations between transition systems which assure that certain aspects of the behaviour of the systems are the same in a related pair. For many applications it is not possible to maintain such an equivalence unless non-observable (stuttering) behaviour is ignored. However, existing bisimulation relations which permit the removal of non-observable behaviour are unable to preserve temporal logic formulas referring to the next step operator. In this paper we propose a novel bisimulation relation, called next-preserving branching bisimulation, which accomplishes this, maintaining the validity of formulas with the next step, while still allowing non-observable behaviour to be reduced. Based on van Glabbeek and Weijland's notion of branching bisimulation with explicit divergence, we define the novel relation for which we prove the preservation of full CTL*.

As an example for its application we show how this definition gives rise to an advanced slicing procedure for temporal logics, a technique in which a system model is reduced to a slice which can be used as a substitute in verification and debugging. The result is a novel procedure for generating a slice that is next-preserving branching bisimilar to the original model. Hence, we can assure that all temporal logic properties are preserved in the slice and consequently the verification on the slice is sound.

3.6 Tooling Support for Formal Specification Languages

Alexander Raschke (Universität Ulm, DE) and Marcel Dausend (Universität Ulm, DE)

In our talk, we emphasised the creation of specifications. There are plenty of tools processing formal specifications, e.g. theorem provers, model checkers, simulators, but most specification languages are poorly supported during the modelling process itself. As opposed to this, programming languages provide modern IDEs which significantly improve the performance of creating, changing and understanding code. Especially features like e.g. autocompletion, debugging facilities, and refactoring support can help inexperienced users to write their requirements in a formal way. Also, the performance of expert users writing complex specifications can be improved. The purpose of the specification determines the set of possible supporting functions offered to the user. We identified at least three (possibly overlapping) purposes: execution, safety and security analysis, and rapid prototyping. For each purpose, different utilities that probably depend on each other can be offered. For any specification language, typical functions for writing assistance (e.g. auto- completion, quick-fixes), comfortable navigation (e.g. cross-reference, hypertext comments), static analysis (e.g. dead code and unused parameter detection), and graphical representation of the overall text structure are helpful. Users who execute specifications, can be supported by stepwise execution with logging, saving and resuming of system states, and interactive visualisation of executions. For safety and security analysis, existing tools for model checking and theorem proving can be integrated into the development environment. Rapid prototyping based on specifications can be supported by simple yet powerful GUI integration and execution statistics for profiling. The talk concluded with a presentation of the CoreASM tool that focuses on the execution of abstract state machines and already supports some of the presented features for enhancing the creation of specifications. Finally, we suggest to step up the common effort towards integrated tools for specification languages, especially taking into account abstractions, refinement, and integration of verification tools, like model checkers or theorem provers. The main goal of this effort should be to ease the overall process of formal methods application.

4 Working Groups: Distributed Termination Detection (DTD)

4.1 An Executable CoreASM Model of the Termination Detection Protocol

Marcel Dausend, Vincenzo Gervasi, Alexander Raschke, and Hamed Yaghoubi Shahir

Based on a literal translation of the English text into an ASM specification [1], we developed an executable specification in CoreASM. In order to achieve this goal, we have had to tackle the problems mentioned in [1]. We do not repeat the complete ASM model in this abstract. Instead, we focus on the details which have to be changed to make the ASM model executable and invite the reader to read [1] in advance.

The execution of the original translated specification resulted after a few steps in an inconsistent update set (as predicted in [1], Sect. 3), because of a simultaneous update of the current state of a machine to *active* and *passive*.

This happens, when a machine decides spontaneously to become passive although it received a message resulting in an activation. This issue was fixed determining that a machine is only allowed to become (spontaneously) passive if it does not receive a message in the same step. The first and the fourth mentioned problems in [1] can be solved by a special treatment of the master machine m_0. The colour of the master machine does not influence the colour of the token, but the master machine always sends a white token at the beginning of a new probe. For this different behaviour it is necessary to adapt the rule `PassToken` such that the `colorT` is possibly set to black, only if the current owner of the token is not the master machine itself. Additionally, the changed translation of $Rule_0$ considers that `PassToken` is only called if the current machine is not the master. With these changes, it is no more possible, that an inconsistent update set occurs for `colorT`.

The problem with the location `colorM` can be solved by introducing a new constraint in $Rule_{3+4}$ that the master can start a new probe by passing a token only if it is passive.

The inconsistent update set mentioned in problem two can be solved adding one new condition that prohibits sending messages to oneself. This solution also exploits a special property of asynchronous multi-agent ASMs, mentioned in [2, page 209]: "Let X be a finite initial segment of a run of an async ASM. All linearizations of X yield runs with the same final state." This means, all linearizations of runs that would result in an inconsistent update set are discarded. Thus, the execution of an async ASM will automatically avoid all cases, where `hasMessage(..)` is set and cleared simultaneously from different agents. Therefore, it is sufficient to avoid the case, when an inconsistent update set is generated within one agent.

In summary, four of the six mentioned problems could be solved by introducing some conditions and a special treatment of the master machine. Problem 5 could be solved by using queues, a background data structure which is provided by CoreASM, but this approach was – as well as problem 6 – not further examined. The (stepwise) execution of the model helped us identifying the problems and finding solutions for them.

References

1 Vincenzo Gervasi, Elvinia Riccobene, *From English to ASM: On the process of deriving a formal specification from a natural language one.* Dagstuhl Report of Seminar 13372, 2013.

2 Egon Börger, Robert Stärk, *Abstract State Machines.* Springer Verlag, 2003.

4.2 From English to ASM: On the process of deriving a formal specification from a natural language one

Vincenzo Gervasi and Elvinia Riccobene

In this presentation, we traced the model synthesis of an ASM model, starting *literally* from Dijkstra et al.'s text describing an algorithm for distributed protocol termination [2].

In particular, we showed how the description from Dijkstra's paper, if taken at face value, would lead to an inconsistent implementation. We reflect on how a person producing a formal model might be tempted to "correct" the requirements, at times without realising it, in the process of building the model. The net result in such cases is a model which passes verification, but is not validated: thus proving that the wrong system is correct.

4.2.1 Introduction

The first task set by the organisers of Dagstuhl Seminar 13372 on the participants was to build a formal model of a distributed termination algorithm proposed in [2], where it is described in rigorous, but plain English, and try to prove certain desirable properties, such as deadlock-freeness, convergence, etc.

As a first contribution, we decided to focus on the *translation process* (from natural language to a formal model, in this case using the Abstract State Machines formalism), rather than on the *resulting model.* Our aim was to surface instances of *reader fill-in*, i.e. when the specifier, rather than faithfully translating the author's description to a more precise language, tends to fill in missing details, or make assumptions that are not explicitly stated in the source text, or force his or her particular interpretation on the author's words.

Often, this phenomenon is desirable: in that a competent specifier, knowing that the author's intention *must have been* to specify a correct algorithm, is justified in establishing any missing assumption or in choosing that particular interpretation which makes the formal model of the system correct. However, in many other cases there is a huge risk associated to this behaviour, in particular where the subject matter is not "pure" computer science (about which a specifier will probably know much), but some other domain (about which a specifier might well know little).

Our method has thus been to translate as faithfully as possible Dijkstra's English to Abstract State Machines rules, and then reflect on the shortcomings and problems with such a model.

4.2.2 Building a model

In our presentation, selected excerpts from [2] are presented side-by-side their ASM counterpart. We will consider the relevant bits of information in textual order, based on their occurrence in the paper, and later use the composition features of ASMs to collate all the rules in a single coherent model.

We start with the definition of machines and their state

We consider N machines, each of which is either active or passive.	$Machines \equiv \{m_0, ..., m_{N-1}\}$ $state : Machines \rightarrow \{active, passive\}$

The next fragment tells us that message transmission is instantaneous: that is, as soon as a machine sends a message (which we define as a macro), the message is available for the recipient to process. Notice we need not formalise the restriction that only active machines send messages (we could, for example, use an **if** clause) as we interpret this as a property that has to be verified (namely, that a non-active machine will not attempt to SENDMESSAGE).

Only active machines send messages to other machines; message transmission is considered instantaneous.	$\textsc{SendMessage}(dest) \equiv$ $\quad hasMessage(dest) := true$

Machines switch from active to passive state and vice versa according to the following two rules:

After having received a message, a machine is active. The receipt of a message is the only mechanism that triggers for a passive machine its transition to activity.	$\begin{array}{l} \textsc{Machine} \equiv \\ \quad \dots \\ \quad \textbf{if } hasMessage(self) \\ \qquad state(self) := active \\ \quad \dots \end{array}$
For each machine, the transition from the active to the passive state may occur “spontaneously”.	$\begin{array}{l} \textsc{Machine} \equiv \\ \quad \dots \\ \quad \textbf{choose } spontaneous \textbf{ in } \{true, false\} \textbf{ do} \\ \qquad \textbf{if } state(self) = active \textbf{ and } spontaneous \\ \qquad\quad state(self) := passive \\ \quad \dots \end{array}$

Again we have above a negative clause, that we can verify by inspecting our machine to check that no other assignment $state(.) := active$ occurs beyond the one above. Notice that it is not clear from the text whether being passive is a prerequisite for reception of a message (probably not); this might potentially be a property for verification. On the contrary, the spontaneous transition from active to passive can only happen if a machine is in the active state.

The text continues by making it explicit what a stable state is in such a system; we model the same concept with a derived function *stable*.

From the above it follows that the state in which all machines are passive is stable.	$stable \equiv \textbf{forall } m \textbf{ in } machines, state(m) = passive$

The main property for the correctness of the algorithm (namely: that it does indeed detect termination) is stated as follows, and does not correspond to explicit ASM rules (but rather, to a condition that has to be proved):

The purpose of the algorithm to be designed is to enable one of the machines, machine nr. 0 say, to detect that this stable state has been reached.	How a machine can “detect” termination will be defined later.

The paper proceeds by stating initial conditions, both in terms of initial states of the various machines, and of topology (of which, we consider only the *token ring* scheme that is used afterwards in the paper).

It is furthermore required that the detection algorithm can cope with any distribution of the activity at the moment machine nr. 0 initiates the detection algorithm.	$\begin{array}{l} \textsc{Init} \equiv \\ \quad master = m_0 \\ \quad \textbf{forall } m \textbf{ in } Machines \textbf{ do} \\ \qquad state(m) := \textbf{choose in } \{active, passive\} \end{array}$
Two orderly configurations present themselves [...] the N machines arranged in a ring.	$\begin{array}{l} pred, succ : Machines \rightarrow Machines \\ pred(m_i) \equiv m_{(i-1) \bmod N} \\ succ(m_i) \equiv m_{(i+1) \bmod N} \end{array}$

Finally, the paper introduces the *token* that is passed, according to the ring topology, from machine to machine – and which will be a determining factor in detecting termination.

We assume the availability of communication facilities such that [. . .]	$\begin{array}{l} \textsc{PassToken} \equiv \\ \quad hasToken(self) := false \\ \quad hasToken(pred(self)) := true \end{array}$
The token being returned to machine nr. 0 will be an essential component of the justification that all machines are passive.	$\begin{array}{l} \textsc{Machine} \equiv \\ \quad \ldots \\ \quad \textbf{if } (self = master) \textbf{ and } tokenIndicatesFinished \\ \qquad done := true \end{array}$

We must thus prove, *done* $\implies$ *stable*. Notice that *stable* could hold even if the *master* (that is, machine nr. 0) hasn't noticed it yet, so *stable* $\implies$ *done* is not needed, nor requested.

Finally, the paper describes various "rules", that are in effect the algorithm itself. Rule 0 describes the circumstances under which the token is passed around.

Rule 0. When active, machine nr. i+1 keeps the token; when passive, it hands over the token to machine nr. i	$\begin{array}{l} \textsc{Rule}_0 \equiv \\ \quad \textbf{if } hasToken(self) \\ \qquad \textbf{if } state(self) = active \\ \qquad\quad \textbf{skip} \\ \qquad \textbf{else} \\ \qquad\quad \textsc{PassToken} \end{array}$
Next to be taken into account is the possibility of messages being sent: [...]	$\begin{array}{l} \textsc{Machine} \equiv \\ \quad \ldots \\ \quad \textbf{choose } spontaneous \textbf{ in } \{true, false\} \textbf{ do} \\ \qquad \textbf{if } state(self) = active \textbf{ and } spontaneous \\ \qquad\quad \textbf{choose } dest \textbf{ in } machines \textbf{ do} \\ \qquad\qquad \textsc{SendMessage}(dest) \end{array}$
To this end each machine is postulated to be either black or white.	$color_M : Machines \rightarrow \{black, white\}$

Notice that in the first fragment, the condition **if** *hasToken*(*self*) is implied by the use of the verb "keeps" in the original (but it would be wise, in a real case, to inquire with the author of the original text).

The next rule details how machines change their colour, and introduce the token's colour:

Rule 1. A machine sending a message to a recipient with a number higher than its own makes itself black.	$\begin{array}{l} \textsc{SendMessage}(dest) \equiv \\ \quad \ldots \\ \quad \textbf{if } (dest >_M self) \text{ /* relaxed in Rule 1' */} \\ \qquad color_M(self) := black \end{array}$
To this end the token is postulated to be either black or white.	$color_T :\rightarrow \{black, white\}$

Coloring of the token is the subject of rule 3, where the concept of a *probe* is also introduced.

Rule 2. When machine nr. i+1 propagates the probe, it hands over a black token to machine nr. i if it is black itself, whereas while being white it leaves the colour of the token unchanged.	PassToken ≡ … **if** $color_M(self) = black$ $color_T := black$
[...] unsuccessful probe: when a black token is returned to machine nr. 0 or the token is returned to a black machine nr. 0, the conclusion of termination cannot be drawn.	$unsuccessful \equiv$ $self = master \wedge hasToken(self) \wedge (color_T = black \vee color_M(self) = black)$

We will model rules 3 and 4 together, as they jointly describe how a probe is initiated.

Rule 3. After the completion of an unsuccessful probe, machine nr. 0 initiates the next probe. **Rule 4.** Machine nr. 0 initiates the probe by making itself white and sending a white token to machine nr. N-1	Rule$_{3+4}$ ≡ **if** $(self = master)$ **and** $unsuccessful$ $color_M(self) := white$ $color_T := white$ PassToken

The last rule details how passing the token changes the colour of a machine:

Rule 5. Upon transmission of the token to machine nr. i, machine nr. i+1 becomes white. (Note that its original colour may have influenced the colour of the token).	PassToken ≡ … $color_M(self) := white$

The final specification was obtained by collating all the fragments we have described above, and adding some additional initialisation to assign initial colours to the various machines and the token, to create ASM agents and assign their programs so that each would execute (in a truly distributed, concurrent fashion) the behaviour specified for the algorithm's machines. Space considerations prevent us from listing here the final specification, and its CoreASM [4] executable equivalent (which, anyway, differs only in minor syntactic ways from what we have shown above).

4.2.3 Problems with the model

We purposely ignored many problems in the direct translation, adhering thus to the position advocated by Berry in [**?**] than an "ignorant" mindset facilitates exposing errors in translating from requirements to formal models.

In fact, our directly translated model has exposed several problems and assumptions[1], that we can only briefly list here:

1. Rule$_2$ and Rule$_{3+4}$ might be executed in parallel, causing a conflicting update to $color_T$.
2. The paper does not explicitly specify when or if a message is consumed. We can infer that it will be consumed upon the machine transitioning to *active*, by adding

[1] Some of these have been revealed by direct inspection of the model, others have been exposed in simulating the execution in CoreASM.

$hasMessage(self) = false$ after $state(self) := active$, but that might cause additional problems (namely, if in the same step a machine m transitions to active and another machine m' sends a message to m, we would have an inconsistent update to $hasMessage(m)$).

3. The "spontaneous" transition from *active* to *passive* needs to be restricted by additional conditions, not mentioned in the paper. Namely, if in a step a machine that is already *active* receives a new message and spontaneously transitions to *passive*, we would have again an inconsistent update.
4. Our macros SendMessage and PassToken may cause an inconsistent update to $color_M(m)$ if m "spontaneously" decides to send a message to any other machine m' in the same step in which it is passing the token to the next machine in the loop.
5. Multiple agents could send messages to the same machine in the same step (true concurrency). We do not know if multiple messages gets collapsed into a single activation, or enqueued and then slowly consumed in subsequent steps, etc.
6. Messages in transit: while message passing is instantaneous, activation of machines might be not. In other words, it may happen that a message is delivered to a machine immediately (i.e.: in the next immediate step) after being sent, but the agent executing that machine will react to the message with a certain delay. The algorithm is correct if activation is also instantaneous, but while the text explicitly states that message delivery is instantaneous, it does not specify that machines are infinitely fast[2].

It will be apparent that many of these problematic issues are exposed by the fully parallel and concurrent nature of distributed ASMs. Any purely sequential or interleaving model would not reveal many of these problems, and would not be faithful to the kind of distributed system imagined by the authors of the original text.

4.2.4 Conclusions

Our intention for the first session of the Seminar was to focus on using ASMs for validation purposes, rather than for verification. Indeed, our attempt at direct translation of the English text from [2] has exposed a number of issues which, if only it were still possible, we would have liked to discuss with the author of our requirements, in a typical instance of the virtuous circle where difficulties with the formal specification prompt further elicitation of the requirements.

After the Seminar we became aware that an ASM model for the same algorithm, completed with a formal verification, had already been published well over a decade ago [3]. In that work, the authors stress mathematical verification, relying on an informal justification for the validation part. We intend to compare and contrast our approach to validation with the models built for verification in [3] as part of future work.

References

1 D. Berry. The importance of ignorance in requirements engineering. *Journal of Systems and Software*, 28(1):179–184, 1995.

2 E. W. Dijkstra, W. Feijen, and A. van Gasteren. Derivation of a termination detection algorithm for distributed computations. *Information Processing Letters*, 16(5):217–219, 1983.

[2] We believe that any concurrent model where the *sender* of a message alters the internal state of the *receiver* in the same atomic action is not representing faithfully the distributed nature of the system.

3 R. Eschbach. A termination detection algorithm: specification and verification. In J. Wing, J. Woodcock, and J. Davies, editors, *FM'99 – Formal Methods*, volume 1709 of *Lecture Notes in Computer Science*, pages 1720–1737. Springer Berlin Heidelberg, 1999.
4 R. Farahbod, V. Gervasi, and U. Glässer. CoreASM: An extensible ASM execution engine. *Fundamenta Informaticae*, 77:71–103, Mar./Apr. 2007.

4.3 Modelling and Validation of the Dijkstra's Case Study through the ASMETA Framework

Paolo Arcaini and Angelo Gargantini

We present the modelling and validation of the case study *Derivation of a termination detection algorithm for distributed computations* by Dijkstra [1], through the ASMETA framework [2]. We have implemented in AsmetaL [3], a concrete syntax for ASMs, the high-level model of the case study presented in the abstract of Gervasi-Riccobene and reflecting "literally" the requirements of the algorithm. By simulation and model review [7] of the AsmetaL model, we have highlighted some ambiguities of the requirements. Model review, in particular, has proved to be an affective *push-button* validation technique. The aim of model review is to determine if a model is of sufficient quality; it allows to identify defects early in the system development, reducing the cost of fixing them. The *AsmetaMA* tool [4] permits to perform *automatic* review of ASMs; it looks for typical vulnerabilities and defects a developer can introduce during the modelling activity using the ASMs.

The application of model review to the AsmetaL model has permitted to identify several inconsistent updates that could rise during simulation. Such inconsistent updates derived from simultaneous behaviours that were not properly guarded in the model.

References

1 E. W. Dijkstra, W. Feijen, and A. van Gasteren. Derivation of a termination detection algorithm for distributed computations. *Information Processing Letters*, 16(5):217–219, 1983.
2 The ASMETA website. http://asmeta.sourceforge.net/, 2013.
3 A. Gargantini, E. Riccobene, and P. Scandurra. A Metamodel-based Language and a Simulation Engine for Abstract State Machines. *J. Universal Computer Science*, 14(12):1949–1983, 2008.
4 P. Arcaini, A. Gargantini, and E. Riccobene. Automatic Review of Abstract State Machines by Meta Property Verification. In C. Muñoz, editor, *Proceedings of the Second NASA Formal Methods Symposium (NFM 2010)*, pages 4–13. NASA, 2010.

4.4 A VDM model of the Termination Detection Protocol

Jens Bendisposto, Egon Börger, Ian J. Hayes, Peter Gorm Larsen, and Andreas Prinz

It was tried to model the system in two ways. In a first attempt a functional specification style was attempted. This did not produce a satisfactory result due to the non-determinism in the protocol that lead to a too complex model. In a second attempt a state-based approach was followed which produced a satisfactory model, in particular, dealing well with the non-determinism of the protocol.

4.5 B and Event-B Models for Distributed Termination

David Deharbe, Marc Frappier, Thierry Lecomte, Michael Leuschel, and Laurent Voisin

In the talk we presented the results of our working group on developing B and Event-B versions of Dijkstra's distributed termination algorithm. In the working group, we first decided upon the global system view and the events of the systems, and then gradually defined the events and introduced variables as needed. Refinement and invariants were added later. Then an Event-B model in Rodin and a classical B model were developed in parallel. In the end, we managed to prove the invariants of the Event-B model using Rodin and the invariants of the classical B model using Atelier-B. We used ProB to validate the absence of deadlocks and various LTL properties. The proper functioning of the models was checked by graphical animation using BMotionStudio. Various errors were uncovered by both model checking and animation.

4.6 Modeling Dijkstra's Termination Detection Algorithm in TLA$^+$

Leo Freitas, Stefan Hallerstede, Dominik Hansen, Markus A. Kuppe, Fernando Mejia, Stephan Merz, Hernán Vanzetto, and Kirsten Winter

TLA$^+$ [4] is a formal specification language that is mainly intended for distributed algorithms. It is based on untyped Zermelo-Frankel set theory for modelling the data structures and on the Temporal Logic of Actions for specifying the possible executions. Typically, TLA$^+$ specifications define transition systems, represented by a state predicate defining the initial states, a transition predicate delimiting the next-state relation, and fairness conditions.

On the first day of the seminar, we modelled Dijkstra's termination detection algorithm [2] in TLA$^+$. We started by determining the representation of the system state by two variables indicating the state of every node (its status can be active or passive, its colour can be white or black) and of the token (its current position and its colour). The definitions of the initial condition and of the next-state relation follow quite directly Dijkstra's description. Having specified the state we formalised the behaviour of the system in terms of a predicative transition relation. We found that a discussion of the formal specification of the algorithm helped us clarify our understanding of it and solve problems in a first version without even resorting to formal verification. Interesting properties of the protocol were then specified in temporal logic.

We also noted a distinction in modelling style: some group members prefer directly giving definitions of algorithm operations (represented as TLA$^+$ actions), whereas others first think about state invariants. Arguably, the former is more dynamic and enables for testing the model using automated tools, whereas the latter builds a solid understanding of the reasons for the algorithm being correct. In practice, it is hard to clearly differentiate this question of style from a (perhaps unconscious) bias towards the functioning of the tools associated with the method.

Tool support for TLA$^+$ comes through the TLA$^+$ Toolbox, an IDE for developing and analysing TLA$^+$ specifications. In particular, the model checker TLC confirmed that the previously defined correctness properties hold for fixed finite instances of the algorithm. In particular, all nodes are indeed inactive when the algorithm detects termination, and the algorithm will eventually detect termination when all nodes are inactive.

The TLA$^+$ proof system TLAPS [?] provides assistance for carrying out interactive proofs and verifying properties of TLA$^+$ specifications. TLA$^+$ contains a hierarchical language for writing proofs that TLAPS checks with the help of different back-end provers, including the tableau prover Zenon, an encoding of TLA$^+$ in the logical framework Isabelle, and a generic interface to SMT solvers. We first used TLAPS to prove a simple invariant that expresses type correctness. Dijkstra's paper also states an invariant that should be satisfied by the algorithm. To our surprise we were at first unable to prove that invariant, and in fact it did not hold: we had developed a variant of Dijkstra's algorithm. We were able to restate the invariant and achieve a correctness proof of the algorithm with the help of the modified invariant. Liveness of the algorithm cannot currently be proved with the help of TLAPS, as it does not yet support proofs in temporal logic.

The TLA$^+$ model of the algorithm can be translated to B [?], and this enables the analysis and visualization of the model using ProB [5]. It would be beneficial to integrate this translation and ProB in the TLA$^+$ Toolbox.

References

1 D. Cousineau, D. Doligez, L. Lamport, S. Merz, D. Ricketts, H. Vanzetto. TLA$^+$ Proofs. 18th Int'l Symp. Formal Methods (FM'12), LNCS, Vol. 7436, pp. 147-154, Springer, 2012.

2 E. W. Dijkstra, W. H. J. Feijen, A. J. M. van Gasteren. Derivation of a termination detection algorithm for distributed computations. Inf. Proc. Letters 16:217-219 (1983).

3 D. Hansen, M. Leuschel. Translating TLA$^+$ to B for Validation with ProB. J. Derrick, S. Gnesi, D. Latella, H. Treharne (eds.): 9th Intl. Conf. Integrated Formal Methods (iFM 2012). Springer LNCS 7321, pp. 24-38. Pisa, Italy, 2012.

4 L. Lamport. Specifying Systems. Addison Wesley. Boston, Mass., 2012. See also http://research.microsoft.com/en-us/um/people/lamport/tla/tla.html.

5 M. Leuschel, M. J. Butler. ProB: an automated analysis toolset for the B method. STTT 10(2):185-203 (2008).

5 Working Groups: Automated Teller Machine (ATM)

5.1 The Integrated Use of Event-B and ASM via UML-B

Paolo Arcaini, Elvinia Riccobene, and Colin F. Snook

In this talk, we discussed the possibility to integrate the use of Event-B and ASMs (Abstract State Machines) for model development and analysis. The two methods provide different models of computation and methods of analysis, so they can be applied for different verification and validation purposes. The "Cash-point Service" case study was used to show the feasibility of the integrated use of the two formal methods. We discussed the validity of the UML-B representation as a common basis for both target notations, and an UML-B model of the case study was developed in terms of class diagrams and hierarchical state machines. Then parallel Event-B and ASM models were derived, even if not in a complete way, from the UML-B model for different analysis purposes: correct refinement proofs could be achieved by exploiting the Rodin tool support [1] for proving Event-B models; model simulation, model review and model checking could be performed by using the ASMETA toolset [2] for ASM models. The future intention is to automatise the derivation of ASM and Event-B models from UML-B, and to ensure that the Event-B and ASM models are equivalent despite differences in the semantics of the notations. We also want to experiment further advantages coming from the integrated use of the analysis techniques of the two methods.

References

1 The RODIN website. http://www.event-b.org/, 2013.

2 The ASMETA website. http://asmeta.sourceforge.net/, 2013.

5.2 Specification of the Cash Dispenser in Parallel using ASM and VDM

Cliff B. Jones, Peter Gorm Larsen, Andreas Prinz, Alexander Raschke, and Colin F. Snook

5.2.1 On the VDM model

The purpose of our first model is to make a quick analysis of the main functional requirements, essentially ignoring concurrency issues. Thus, this model describes aspects of the central resource and only one till. Once the basic functionality is better understood we start to consider multiple tills. The model is described in VDM-SL as a short, flat specification. This enables abstraction from design considerations and ensures maximum focus on high-level, precise and systematic analysis.

We develop an abstract model in VDM-SL in order to clarify and analyse requirements. This model is validated using a testing approach. The purpose of our models is to analyse the main functional aspects of the system, including security aspects related to misuse of cards. For example, the models treat the daily limit policy, the validation of PIN codes, limits on PIN code attacks, and reports of illegal (e.g. stolen) cards. However, we abstract away from, for example, the choice of concrete databases, communication protocols and encoding of PIN codes.

5.2.2 Comparison

Some "correspondences" are easy to spot:

VDM	ASM
state definition explicit	implied/extracted
state components	"controlled functions"
invariants	tool dependant
pre	written as if statements
post condition (not used in this case)	only explicit
records	via (selector) functions
not applicable; records are values	record creation is bound to be a step
pattern match on records	not available
other "sophisticated" data types	sets/sequences
doesn't have a global let	"derived" functions

ASM and VDM have very different concurrency principles

- In VDM++
  - explicit synchronization
  - Using mutex and permission predicates
  - Closer to implementation language
  - Interleaving concurrency
- In ASM
  - More high-level "magic"
  - Potential problem that implementer "forgets" synchronisation
  - Convenient abstraction for design
  - True concurrency

5.3 Specification of the Cash Dispenser in Parallel using Z and TLA$^+$

Leo Freitas

TLA$^+$focuses on operations, Z on state. Concerning the use of TLA$^+$actions and Z schemas modelling styles are similar.

5.3.1 Differences

- Outputs are handled differently. TLA$^+$uses "state" variables. Z uses schema variables marked with a "!".
- Z preconditions aren't guards.
- Z has complex types yet has untyped logic. TLA$^+$is untyped.
- Updates in Z via promotion, that is local and global. Updates in TLA$^+$are global.
- Z schema calculus
- Verification methods: TLA$^+$uses model checking and proof whereas Z uses proof only (ZEves)

5.4 Specification of the Cash Dispenser in Parallel using ASM and TLA$^+$

Egon Börger, Ian J. Hayes, Fernando Mejia, Stephan Merz, Klaus-Dieter Schewe, and Kirsten Winter

5.4.1 Desiderata

The produced models had to satisfy the following desiderata:

- component-based modelling
- replicate independent ATM components

- focus on observable ATM behaviour
- obtain as similar models as possible
- don't aim for complete specification

5.4.2 Observations

We have made the following observations concerning the two modelling approaches:

- non-interleaving, compositional specifications not supported by TLA$^+$tools
- behaviour specifications are very similar (control ASMs, state machines in TLA$^+$)
- properties stated outside the ASM language (but supported by ASM tools)
- translation from ASM to TLA$^+$should be straightforward

5.5 Specification of the Cash Dispenser in Parallel using ASM and B

Marcel Dausend, Marc Frappier, and Angelo Gargantini

We have modelled the "Cash-point" Service proposed by the organisers in collaboration and in parallel using both methods, Abstract State Machines (the tool Asmeta) and classical B with Atelier B and ProB.

The goal has been twofold. One objective has been to create two models of the case study as a basis for comparison. The subsequent objective was to compare the models' notations and the process of developing those models. For this reason, we have decided to proceed incrementally, keeping the names of variables, events and rules as close as possible in the two models. In order to guarantee that both models capture the same behaviour, we systematically performed simulation and comparison of the two models. In this way the ASM people could learn some details of the syntax and semantics of B and vice-versa.

We were able to identify some shared characteristics between ASM and B constructs. B sets and their properties are mapped to ASM (abstract) domains and their initial elements. B states correspond to (dynamic controlled) functions and variables of ASMs. Constants can be easily represented in both notations. Refinement has been used in the initialisation in B to guide ProB for the simulation of the nondeterministic abstract initialisation. In ASMs it has been done by function definition and state initialisation. ASM rules and B operations specify how the state evolves. We could map every B operation to one ASM rule. Preconditions of B operations are mapped in ASMs to guards of conditional rules, while B substitutions are mapped to ASM function updates. Both formalisms allow explicit use of variables set by the environment. These variables are normally user inputs during simulation. In B user inputs are parameters in operations, while in ASM they are modelled as monitored functions. In B all the operations are enabled (if their preconditions are true), while in ASM (at least in the used tool) the user has to write a main rule that explicitly runs all the rules in parallel.

6 Working Groups: Comparison of Methods and Tools

6.1 Summary of Meeting of Workgroup on Methodology

Albert Fleischmann, Uwe Glässer, Ian J. Hayes, Stefan Hallerstede, Dominik Hansen, Laurent Voisin, and Kirsten Winter

The discussion in the meeting proceeded in two stages. In the first stage we tried to delineate our understanding of modelling and its purpose as it would be understood by the different communities present at the seminar.

6.1.1 Stage 1

The purpose of modelling is System Engineering rather than (pure) Software Engineering. This position has consequences on the scope of problems to which modelling is applied ranging from (1) system requirements analysis to (2) system specification and verification to (3) concrete systems and software exploitation.

The 3 problem domains have different techniques associate with them. Usually tools do not focus on more than one domain. The focus on different domains may explain some differences in notation. (But often notations seem to evolve haphazard driven by the problems they were first applied to.)

Concerning (1) common techniques are
- (partial) model building
- traceability to requirements, needs and objectives
- animation
- simulation
- integrity analysis (details vary)

Concerning (2) common techniques are
- safety/liveness analysis
- traceability to requirements and statements in the formal model
- model checking, theorem proving, refinement, abstraction
- simulation

Concerning (3) common techniques are
- (program) execution
- model-based texting
- runtime verification
- traceability to requirements and statements in the formal model

Given the variation in the application domains of the different methods and tools it does not appear reasonable to attempt to unify them. But it would be useful to find ways to achieve tight collaboration in order to address problems that span several of the domains.

6.1.2 Stage 2

In the second stage we formulated some recommendations that could help achieving such collaborations.

6.1.2.1 Recommendation A

"Compatible" specification notation dialects shall be used that permit switching between methods and tools during development. The meaning of "compatible" is not well-understood. A possible meaning could be "only requires simple translation". This would coincide with the finding of the working group on abstract syntax. The different methods should co-operate in particular with respect to important modelling concepts such as concurrency and distribution.

6.1.2.2 Recommendation B

Transfer of concepts between methods should be encouraged, e.g.,
- modules from ASM to Event-B
- refinement from Event-B to ASM

6.1.2.3 Recommendation C

Challenges could be put forward resulting from the transfer, e.g.,
- use ASM for composing Event-B modules
- use Event-B to prove certain ASM refinements

6.2 Comparison of Methods

Peter Gorm Larsen, Andreas Prinz, Colin F. Snook, and Hamed Yaghoubi Shahir

In the discussion, we agreed that all the methods are state-based and have a semantically similar idea of state. The same is true for the initial state. Moreover, all the methods use some kind of state transitions. There is not much alignment related to the notation of concepts and the (detailed) semantics of concepts. This leads to problems with different semantics for visually the same concepts and the same semantics for visually different concepts. The main difference is often the development process used, and the example we had showed this clearly. In our group, we worked with functional formulation first, which did not work out too well. Afterwards, a more relational approach was followed making our solution more similar to the other solutions.

6.3 Defining ASMs as Event-B Machines

Egon Börger and Laurent Voisin

6.3.1 Abstract State Machines

A *(sequential) ASM M* is a set of rules of form

$$\textbf{if } \mathit{cond} \textbf{ then } \mathit{Updates}$$

where *cond* is a first-order (typically set theory) formula and *Updates* is a set of function updates $f(t_1, \ldots, t_n) := t$ with parameterised locations (array variables) $(f, (val_1, \ldots, val_n))$ and terms $t_1, \ldots, t_n, t$. It is allowed to **choose** among or to generalise **forall** elements of a set in a rule. Also other usual notations like **if then else**, **let**, etc. and calling (recursive) sub-machines are allowed.

The states of an ASM are (Tarski) structures with static and dynamic part. In each *step* M executes all its rules, performing simultaneously all the updates of each of its rules whose *cond* in this state is true.

Input is taken as part of the initialisation of a machine M and after each machine step by a step of the *environment* which updates the so-called monitored locations of the machine state. For shared locations a protocol is required to avoid conflicting updates.

An *(async) ASM* is a set of agents each equipped with a sequential ASM.

6.3.2 Event-B Machines

Except for notational differences Event-B machines have the same states as ASMs, to be precise structures of a given signature with a static part (called 'context') of sets s ('universes'), constants c, properties(c, s) ('axioms' and 'theorems' to be proved within the context) and a dynamic part of (pairwise distinct) variables v, invariants (predicates the variables must be proved to satisfy in every state that is reachable from an initial state) and the environment which is part of the model (closed system).

Inputting is done by non-determinacy. The initialisation is described via a special event with guard true.

Events are without loss of generality instances of the following general form (stated in ASM notation):

```
if forsome $p$  $P(p, v)$ then
  choose $p$ with $P(p, v)$  in $action(p, v)$
where
  $action(p, v) =$ choose $v'$ with $Q(v, v', p)$
    $v := v'$ // NB.$v, v'$ generally are vectors
```

A step of an Event-B machine with a finite number of events $event_i$ $(1 \leq i \leq n)$ can be described in ASM terms as follows:

```
choose $i$ with $guard_i$ in $event_i$ // interleaving view
  where
    $event_i =$ if $guard_i$ then $rule_i$
```

A constraint requires that no parallel update is allowed for the same variable.

6.3.3 Sequential ASMs encoded as Event-B Machines

To simplify the exposition of the coding idea assume that each **choose** construct is replaced by an explicit selection function.

A brute force transformation of (sequential) ASMs rewrites the set of parallel rules in such a way that all truth-functional combinations of rule conditions are listed explicitly and guard the respective function updates. This comes up to rewrite the given ASM into a flat ASM with only rules of form **if** *cond* **then** *Updates* where all guards are pairwise disjoint. As a consequence, one can interpret the resulting ASM semantically as an Event-B machine where in each state only one 'event' can be selected. All non-determinism is hidden in the selection functions.

- replace

```
if $cond_1$ then $Updates_1$
if $cond_2$ then $Updates_2$
```

- by the equivalent machine

```
if $cond_1$ and $cond_2$ then
  $Updates_1$
  $Updates_2$
if $cond_1$ and not $cond_2$ then $Updates_1$
if $cond_2$ and not $cond_1$ then $Updates_2$
```

A more economical encoding goes as follows:

```
if $c_1$ then $f(s_1) := t_1$
if $c_2$ then $f(s_2) := t_2$
```

is encoded by

```
when $c_1$ or $c_2$ then
$f := f \oplus$
  $\{(a, v) \mid c_1$ and $(a, v) = (s_1, t_1)$
    or $c_2$ and $(a, v) = (s_2, t_2)\}$
```

To prove: f is a (partial) function.

Similarly the **forall** construct can be encoded via a set update.

We still have to clarify whether and how to deal with recursive sub-machines and with the **undef** concept in ASMs.

6.3.4 Where the differences are

The main difference is in the refinement notions (see [1] and [2, Ch.3.2.1]). The Event-B concept is tailored to provide as much as possible interaction-free refinement correctness verifications. The ASM refinement concept is driven first of all by modelling concerns to describe design ideas in an accurate and as smooth as possible manner, delegating mathematical verification (where considered to be needed) to a second step.

References

1 J.-R. Abrial. *Modeling in Event-B: System and Software Engineering.* Cambridge University Press, Cambridge, 2010.

2 E. Börger and R. F. Stärk. *Abstract State Machines. A Method for High-Level System Design and Analysis.* Springer, 2003.

6.4 Closing the Gap between Business Process Models and their Implementation: Towards Certified BPMs

Albert Fleischmann and Egon Börger

The gap between on the one side the users' understanding of Business Process Models (BPMs), even if described using standardised languages like BPMN, and on the other side the run behaviour of model implementations is still with us. We explain how Abstract State Machines (ASMs), tailored as a domain specific (to a large extent diagrammatic) modelling language, allow the BP experts to design BPMs with the help of a graphical editor in such a way that the underlying ASM models constitute a reliable precise contract – a contract which guarantees to the BP domain experts that the application-domain focussed understanding of the BPMs they design is also a correct understanding of the code behaviour provided by the implementation of the models by software experts. This opens the way to the development of certifiably correct BPMs and their implementations. We instantiate the claim by ASM models for the behavioural meaning of the graphical notations used in Metasonic's industrial BPM tool suite.

6.5 Comparison of Methods and Tools: the Report of the ASM Group

Elvinia Riccobene, Paolo Arcaini, Marcel Dausend, Albert Fleischmann, Angelo Gargantini, Vincenzo Gervasi, Uwe Glässer, Alexander Raschke, Gerhard Schellhorn, Klaus-Dieter Schewe, Qing Wang, and Hamed Yaghoubi Shahir

This report presents the results of the ASM group discussion, held at the second day of the seminar, regarding the comparison of methods and tools on specific questions and reflection points.

The following questions were arguments of the discussion:

- Why were there variations of the models? Why did you go for one model rather than another? Which modelling process did you use?
- Compare with other models you have seen: Commonalities and Differences.
- List weak points of your method, tools, models.
- How far is the model style pre-determined by the tool or adapted to the tool?
- Wish list for new features, tools, etc.; especially those you have seen from other groups.
- Brainstorm: Ideas and suggestions for Integrating tools and methods; make more concrete suggestions, how they would profit, refer to other tools and methods.

For the Dijkstra et al.'s case study, three ASM models were presented. They all *share* the common operational semantics of the ASMs, but present some *variations* due to the different goals which they were developed for. (i) For *requirement elicitation*, an ASM model was developed simply translating the text of the Dijkstra description in terms of transition rules capturing the behaviour of the system at a very high level of abstraction. This model is not "correct" and "complete". Rather, it tries on purpose to expose errors, ambiguities, or incompleteness in the original text. Correctness and completeness can be reached through an iterative process reasoning on requirements. This high level model capturing the informal requirements is usually called "ground model". The ASM ground model of the Dijkstra et al.'s case study results into an asynchronous multi-agent ASM. (ii) For *model validation*, two ASM models were developed and encoded in the languages of the two simulation engines available for ASMs: the CoreASM and the AsmetaS simulator of the ASMETA framework. These two models can be obtained from the ground model by refinement into the tool languages. They present distributed solutions with the same transition rules, they almost use the same syntax (untyped for CoreASM, typed for Asmeta), but they present different ways to refine rules in order to make the model executable. (iii) For *verification* purposes, a specification in KIV was developed using a global model to facilitate the proof of invariants.

Regarding the *modelling process* usually followed when developing an ASM model, the user starts from the textual description of the informal requirements and tries to develop a Ground Model capturing the intended behaviour of the algorithm or system at a high level of abstraction. Then (s)he refines predicates and macros and makes signature precise and complete. The model can be then validated by the use of simulation engines, and other tools, as the model advisor, can be used to discover inconsistencies, redundancy, etc., helping to fix problems. More complex properties can be later proved by using theorem provers (KIV, PVS) or model checkers (AsmetaSMV, CoreASM2SMV).

Discussing *commonalities and differences* of the ASM method with respect to the other formal methods presented at the seminar, the group agreed on the following *commonalities*: state-based nature of the approach, operational models suitable for animation and verification; and *differences* (commonly intended as advantages of the method): ability to formalise ALL behaviour requirements, and requirements traceability. Both these qualities, not completely present in the other methods, are fundamental to communicate with stakeholders. Among the differences, the necessity was recognised to improve the way to perform model animation that is not supported by the ASM tools at the moment.

As other *weak aspects* of the ASM method and its supporting tools, the group agreed on (i) the absence of a way to express temporal properties (only expressions in CTL/LTL with finite sets are supported); (ii) the lack of integration between different tools for model simulation (CoreASM and Asmeta); (iii) the different ways of data encoding; (iv) the lack of interoperability among encoding into different verification languages (KIV, Promela/SPIN, SMV, PVS).

Regarding the question *if the model style is pre-determined by the tool or adapted to the tool*, the group agreed that modelling style is not constrained by the tools, but a certain adaptation is required to move from the ground model to an executable model, and more effort is required to use verification tools.

On the *wish list* for new features, tools, and other issues seen from other groups, the ASM community agreed on having (i) a higher level notation to facilitate tool integration, and (ii) a way to express temporal features.

On the last question regarding *ideas and suggestions for integrating tools and methods*, all people agreed that is more reasonable and feasible trying to achieve an *integrated use* of different methods and tools, instead of going towards tool integration.

7 Post-Seminar Collaboration Activities

In the sessions on day 3 and day 4 many concrete collaborations and integration efforts have been started. In particular,

- Visualisation & Integration: Jens, Peter, Vincenzo
- UML-B to ASM: Elvinia, Paolo, Colin
- Constraints Solver Integration into VDM: Jens, Peter
- Pro-B Integration into TLA Toolbox and TLC Tool: Dominik, Stephan, Hernan
- CZT to Pro-B (Pro-Z + TLA$^+$): Leo
- Proof GUI: Laurent, Peter
- Destecs Integration: Peter, Jens
- Comparison of Model Checking between ASM and B: Paolo, Michael

- Comparison of Model Development between ASM and B: Marc, Angelo
- Common Language for ASM: Marcel, Alex, Vincenzo, Elvinia, Angelo, Andreas
- Case Study: Uwe, Hamed
- Modelling and Validation of Distributed Situation Analysis (ASM and Event-B): Uwe, Narek, Hamed
- ASM to Event-B Translation: Laurent, Egon

8 Plenum Discussions

The organisers had planned several plenary discussions in a first draft. They had been replaced by plenum discussions by the time the seminar started. The main reason for this was to avoid turning too many participants into "spectators". We believe, this worked well for a small seminar. Plenum discussions may be less effective in large seminars.

In this seminar a plenum discussion was held each time after the groups reunited to present their results. If the time planned for the discussion was not sufficient, the corresponding slot was extended and the programme adapted.

9 Final Programme

Day 1	
09:00–09:30	Opening
09:30–10:30	Short presentation of participants (3 slides per presentations, all presentations in 1 hour!)
10:30–12:00	Modelling case study in groups
12:15–13:30	Lunch
13:30–15:30	Modelling case study in groups
15:30–16:00	Coffee break
16:00–18:00	Presentation of the models to illustrate the different methods and tools
Day 2	
09:00–10:30	Comparison of methods and tools based on the models of day 1
11:00–12:00	Presentation of results of comparison
12:15–13:30	Lunch
13:30–15:30	Discussion continued (45 min) & 3 working groups on integration ($\sim$ 60 min): Methodology, Abstract Syntax, Low-level integration
15:30–16:00	Coffee break
16:00–17:30	Discussion of working group results
17:30–18:00	Talk: Egon Börger: A proposal for including communication into Abstract State Machines
Day 3	
09:00–10:30	Modelling in two formalisms: The FM'99 ATM modelling challenge
11:00–11:45	Talks: Paolo Arcaini & Kirsten Winter
11:45–12:15	Planning of integration sessions for next day
12:15–13:30	Lunch
13:30–17:30	Excursion: A rainy walk

Day 4

09:00–10:00	Wrap up ATM case study of day 3
10:00–12:00	Presentation of solutions of the ATM case study
12:00–12:15	ASM to B: transformation (Egon Börger)
12:15–13:30	Lunch
13:30–15:00	Research and tool talks on various topics (by Kirsten Winter, Laurent Voisin, Marcel Dausend, Leo Freitas)
15:30–17:30	Technical interchange meetings and discussion about architectures between individual groups
17:30–18:00	Talk: Flash File System Verification (Gerhard Schellhorn)
19:30–21:00	Plenum discussion on joint book and other project outcomes (report for Seminar 13372)

Day 5

9:00–11:00	Closing: Conclusions, Proceedings, Book

The rest of the day was left unplanned because many participants left early due to travel arrangements.

Participants

- Paolo Arcaini
University of Bergamo, IT
- Jens Bendisposto
Heinrich-Heine-Universität Düsseldorf, DE
- Egon Börger
University of Pisa, IT
- Marcel Dausend
Universität Ulm, DE
- David Deharbe
Federal University of Rio Grande do Norte, BR
- Albert Fleischmann
Metasonic AG – Pfaffenhofen, DE
- Marc Frappier
University of Sherbrooke, CA
- Leo Freitas
Newcastle University, GB
- Angelo Gargantini
University of Bergamo, IT
- Vincenzo Gervasi
University of Pisa, IT
- Uwe Glässer
Simon Fraser Univ. – B.C., CA
- Stefan Hallerstede
Aarhus University, DK
- Dominik Hansen
Heinrich-Heine-Universität Düsseldorf, DE
- Ian Hayes
The Univ. of Queensland, AU
- Cliff B. Jones
Newcastle University, GB
- Markus Alexander Kuppe
Universität Hamburg, DE
- Peter Gorm Larsen
Aarhus University, DK
- Thierry Lecomte
CLEARSY – Aix-en-Provence, FR
- Michael Leuschel
Heinrich-Heine-Universität Düsseldorf, DE
- Fernando Mejia
Alstom Transport – Saint-Quen, FR
- Stephan Merz
LORIA – Nancy, FR
- Andreas Prinz
Univ. of Agder – Grimstad, NO
- Alexander Raschke
Universität Ulm, DE
- Elvinia Riccobene
University of Milan, IT
- Gerhard Schellhorn
Universität Augsburg, DE
- Klaus-Dieter Schewe
Software Competence Center – Hagenberg, AT
- Colin F. Snook
University of Southampton, GB
- Hernán Vanzetto
INRIA – Nancy – Grand Est, FR
- Laurent Voisin
SYSTEREL Aix en Provence, FR
- Qing Wang
Australian National Univ., AU
- Kirsten Winter
The Univ. of Queensland, AU
- Hamed Yaghoubi Shahir
Simon Fraser Univ. – B.C., CA